DESIGNING THE PHYSICAL DATABASE for SCALABILITY

A UNIFORM, METHODICAL APPROACH

Data Warehouse Summit
Phoenix
Friday, December 11, 1998
9:00 A.M. - 5:00 P.M.

David McGoveran
Alternative Technologies
13150 Highway 9, Suite 123
Boulder Creek, CA 95006
Telephone: 408/338-4621
www.AlternativeTech.com

BEFORE YOU LEAVE…

PLEASE FILL OUT YOUR EVALUATIONS.

Thank you!

ASSUMED BACKGROUND

- **You Understand The Relational Model**
  - **THE BASIC CONCEPTS**
  - **THE DATE-MCGOVERAN DEFINITIONS**
    - » **RELATIONS AND DATABASES**
    - » **RELATION PREDICATES**
    - » **INTEGRITY RULES**
  - **DATA INDEPENDENCE**
    - » **DEFINITION AND IMPORTANCE**

ASSUMED BACKGROUND

- **You Understand Logical Design**
  - **DEPENDENCIES**
  - **NORMALIZATION**
  - **THE DATABASE DESIGN PRINCIPLES**
    - » **THE DATABASE DESIGN PRINCIPLE OF ORTHOGONALITY (MCGOVERAN-DATE)**
    - » **THE DATABASE DESIGN PRINCIPLE OF COMPLETENESS (MCGOVERAN)**
    - » **THE DATABASE DESIGN PRINCIPLE OF MINIMALITY (MCGOVERAN)**
  - **IDENTIFYING PROPER COLLECTIONS OF TABLES**
  - **GUARANTEEING VIEW UPDATABILITY**

THE APPROACH

- **Establish Requirements**
  - **FIXED**
    - » **KNOWN INITIAL SUBJECT MATTER**
    - » **CORRECTNESS**
  - **VARIABLE**
    - » **PERFORMANCE**
    - » **LOAD**
    - » **ADDITIONAL SUBJECT MATTER**
- **Design Goals**
  - **MEET FIXED REQUIREMENTS**
  - **HAVE FLEXIBILITY FOR VARIABLE REQUIREMENTS**

THE APPROACH

- **Technique for Addressing Fixed**
  - **MEET FIXED REQUIREMENTS THROUGH LOGICAL DESIGN**
    - » **INSURE CORRECTNESS**
  - **SEPARATE FIXED FROM VARIABLE THROUGH DATA INDEPENDENCE**
    - » **PERMIT A LARGE CLASS OF PHYSICAL CHANGES**
      - **APPLICATIONS ACCESS ONLY THE LOGICAL SCHEMA**
      - **TOOLS & APPLICATIONS THAT CREATE OR ACCESS THE PHYSICAL SCHEMA LIMIT SCALABIILTY!**
    - » **ACCOMMODATE ADDITIONAL SUBJECT MATTER**
      - ***PROVIDES FUNCTIONAL SCALABILITY*!**

ADDRESSING FIXED THROUGH LOGICAL IS ASSUMED ALREADY DONE IN THIS COURSE

THE APPROACH

- **Technique for Addressing Variables**
  - **ADDRESS VARIABLE GOALS**
    - » **MONITOR / ANTICIPATE LOAD PROFILE CHANGES**
    - » **SELECT / IMPLEMENT BEST PHYSICAL SOLUTION**
    - » **ACCOMMODATE ADDITIONAL SUBJECT MATTER**
      - **EXTEND THE LOGICAL MODEL**
      - **PROPAGATE LOGICAL CHANGES INTO PHYSICAL**

PHYSICAL DESIGN IS A NEVER ENDING ITERATIVE PROCESS . . .

OUTLINE

- The Approach In Overview
- Scalability
  - DEFINITION AND GENERAL GOALS
  - ENABLING PLATFORM ARCHITECTURE SCALE UP
- Data Independence
  - WHAT IS LOGICAL AND WHAT IS PHYSICAL
  - PHYSICAL DERIVED FROM LOGICAL
    - » KEEPING THEM CONSISTENT
  - THE VALUE OF DATA INDEPENDENCE
  - NORMALIZATION VS. DENORMALIZATION
  - STAR SCHEMAS

OUTLINE

- **Load Profiles**
  - **SOURCES OF LOAD**
    - » **FEEDS, REFRESH, LOAD / RELOAD**
    - » **BACKUP, EXTRACTS, STANDARD REPORTS, ADHOC QUERY, MDD / OLAP**
    - » **GROWING USER COMMUNITY**
  - **MEASURING AND SAMPLING**
  - **MODELING**
  - **HANDLING VARIANCE**
- **Getting the Most Out of the Optimizer**
  - **APPLICATION, INDEX, TRANSACTION DESIGN PRINCIPLES**

OUTLINE

- Denormalization
- Missing Information
  - OPTIMIZING FOR SPACE AND PERFORMANCE
- Optimal Storage Management
  - DIMENSIONAL SCHEMAS
  - PRE-AGGREGATION AND SUMMARY TABLES
  - TABLE PARTITIONING
    - » NUMBER, SIZE, AND PARTITIONING METHOD
  - REPLICATION AND OTHER FORMS OF REDUNDANCY
    - » WHEN AND HOW TO USE REPLICATION
- Getting the Most Out of Parallelism
  - TIPS AND TECHNIQUES THROUGHOUT

WHERE WE ARE

- ✓ The Approach In Overview
- Scalability
- Data Independence
- Load Profiles
- Getting the Most Out of the Optimizer
- Denormalization
- Handling Missing Information
- Optimal Storage Management

PART I

SCALABILITY

SCALABILITY

Definition

SCALABILITY IS:

- ***SCALEUP*** **or** ***SPEEDUP*** **(see slides which follow)**
- **WITH RESPECT TO A** ***SPECIFIC RESOURCE MIX***
  - » ***AMOUNT OF MEMORY, NUMBER / SIZE OF STORAGE UNITS, NUMBER OF CPUs, NUMBER OF NODES, et cetera.***
- **OVER A** ***SPECIFIED RANGE***
- **FOR A** ***PARTICULAR WORKLOAD***
  - » ***NUMBER OF USERS, DB SIZE, TRANSACTION RATE, TRANSACTION COMPLEXITY or PROFILE***

- **Conceptual Definition of Speed Up**

  MORE RESOURCES ☑ BETTER PERFORMANCE, SAME WORKLOAD

- **Conceptual Definition of Scale Up**

  MORE RESOURCES ☑ SAME PERFORMANCE, BIGGER WORKLOAD

SPEEDUP
DEFINITION

- **Fixed Workload, More Resources**
  - IMPROVE THROUGHPUT (COMPLETE WORK FASTER)
  - IMPROVE RESPONSE TIME (ENABLE USER MORE QUICKLY)
- ***W* = Work Rate Possible With An Available Resource**
  - *W1 WITH RESOURCE R*
  - *W2 WITH RESOURCE 2 * R*
  - KEEPING OTHER FACTORS CONSTANT
  - MEASURED OVER SAME TIME PERIOD
- ***P* = Percent Speedup**
  - WITH RESPECT TO THE PARTICULAR RESOURCE(S)
  - OVER THE RANGE *R* → *2R*

$$P = (W2 - W1) * 100 / W1$$

SCALEUP
DEFINITION

- **Fixed Elapsed Time, More Resources**
  - INCREASE WORKLOAD (MORE TASKS COMPLETED)
  - IMPROVE THROUGHPUT (FINISH SAME WORK SOONER)
  - IMPROVE CONCURRENCY
- **Resource Required to Perform Fixed Amount of Work**
  - R1 WITH LOAD *L*
  - *R2* WITH LOAD *2 * L*
  - KEEPING OTHER FACTORS CONSTANT
  - MEASURED OVER SAME TIME PERIOD
- ***S* = Percent Scaleup**
  - WITH RESPECT TO THE RESOURCE
  - OVER THE RANGE *L* → *2L*

$$S = (R2 - R1) * 100 / R1$$

SCALABILITY
GENERAL GOALS

The Essence of Scalability is Independence

of. . .

- **COMPONENTS BY FUNCTION AND TASK INSTANCE**
- **RESOURCES ASSIGNED TO INDEPENDENT COMPONENTS**

- **Non-Independence Manifests As. . .**
  - **RESOURCE CONTENTION (WAIT TIME)**
  - **PROCESSING ANOMALIES AND MAINTENANCE SIDE EFFECTS**
  - **INABILITY TO EXPLAIN THE ARCHITECTURE**
  - **INABILITY TO EXPLAIN THE CAUSE OF SYMPTONS**

Avoid These By Building-in Independence

WHERE WE ARE

- ✓ The Approach In Overview
- ✓ Scalability
- Data Independence
- Load Profiles
- Getting the Most Out of the Optimizer
- Denormalization
- Handling Missing Information
- Optimal Storage Management

PART II

DATA INDEPENDENCE

MULTIPLE DATABASE MODELS

- **What is a relational database?**
  - **A COLLECTION OF *FACTS: NOT* A COLLECTION OF *DATA*!**
  - **A FACT IS A TESTABLE RELATIONSHIP AMONG DATA**
- **How many maintain both logical and physical data models?**
- **How many know the distinction?**
- **Who knows the value of the logical model?**
- **Who knows the value of the physical model?**
- **How many maintain a map between them?**

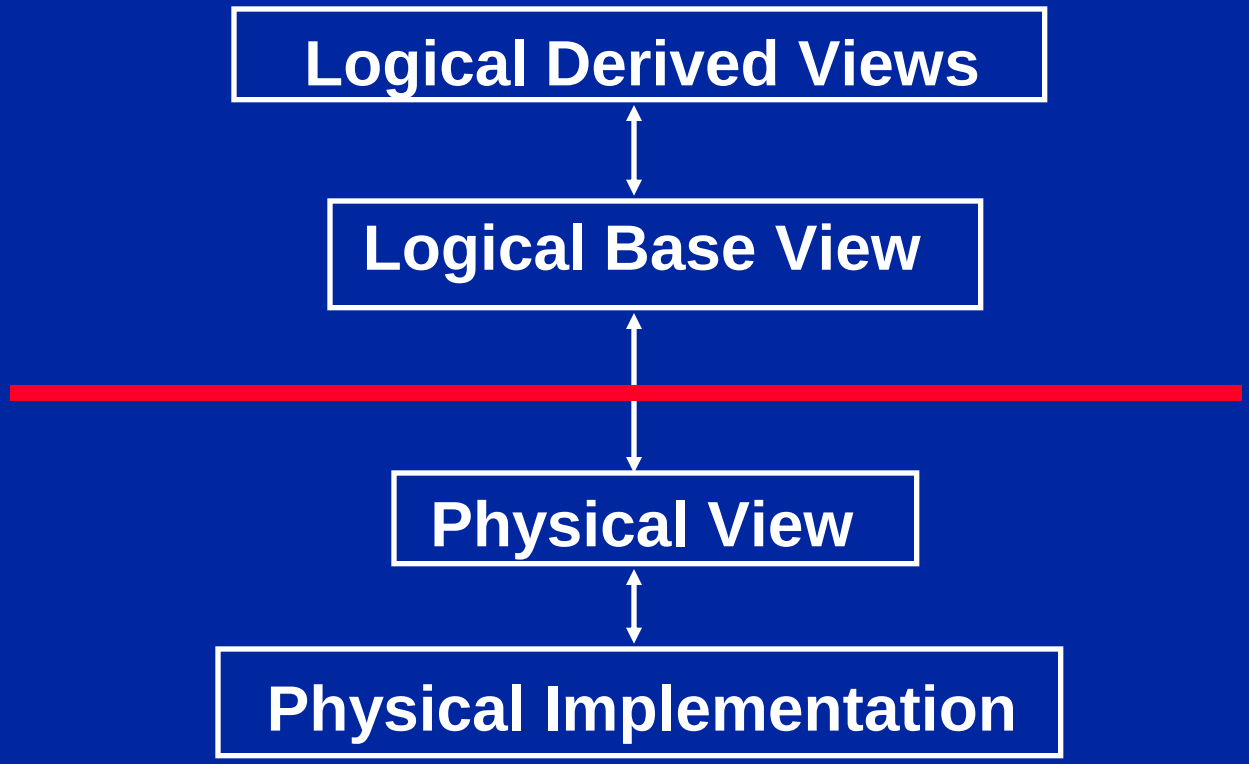
MULTIPLE DATABASE MODELS

- **Why have levels of abstraction?**
  - EASE / SPEED OF DESIGN AND DEVELOPMENT
  - EASE OF MAINTENANCE
  - CORRECTNESS
  - SMALLER, MORE FLEXIBLE SYSTEMS
- **Conceptual**
  - A VIEW OF THE LOGICAL MODEL
  - PRESENTS THE BUSINESS OR CONCEPTUAL VIEW
  - ADDRESSES THE USER'S OR APPLICATIONS VIEW(S)
    - » BOTH PROCESS (BUSINESS TRANSACTIONS) AND DATA
  - CRUCIAL TO EASE OF USE

MULTIPLE DATABASE MODELS

- **Logical**
  - ***GUARANTEES ACCESS (RELATIONAL CORRECTNESS AND COMPLETENESS)***
    - » **BOTH PROCESS (PERMISSIBLE STATE TRANSITIONS) AND DATA**
    - » **A SUCCESSFUL TRANSACTION IS A PERMISSIBLE STATE TRANSITION (TAKES DATABASE FROM ONE CONSISTENT STATE TO ANOTHER)**
- **Physical**
  - ***ADDRESSES EFFICIENCY (PERFORMANCE AND STORAGE)***
    - » **BOTH PROCESS (ACCESS METHODS) AND DATA**
  - **MUST BE A VIEW OF THE LOGICAL MODEL (WHY?)**

LAYERED DESIGN



DATA INDEPENDENCE

- **Logical Mostly Independent of Physical**
  - CAN HIDE STORAGE ALLOCATION AND PERFORMANCE
  - PHYSICAL PLATFORM ISSUES NEED BE KNOWN ONLY TO DBMS
  - SQL ENTANGLES THESE, ESPECIALLY AT TABLE CREATION
- **Applications Access *Only* the Conceptual or Logical Schemas**

Result?

A SCALABLE DESIGN!

- CAN CHANGE THE APPLICATION CODE AND THE PHYSICAL SCHEMA INDEPENDENTLY!
- ADDRESS INVARIANT AND VARIABLE REQUIREMENTS INDEPENDENTLY
- ***ENABLES SCALABLE PLATFORM ARCHITECTURE CHANGES***

WHERE WE ARE

- ✓ The Approach In Overview
- ✓ Scalability
- ✓ Data Independence
- Load Profiles
- Getting the Most Out of the Optimizer
- Denormalization
- Handling Missing Information
- Optimal Storage Management

PART III

LOAD PROFILES

LOAD
WHAT IS IT?

- **Database Load Has Many Factors**
  - **WHAT IS BEING DONE?**
    - » **INPUTS - SEARCH ARGUMENTS (MORE LATER)**
    - » **OUTPUTS - DATA RETRIEVAL AND MODIFICATIONS**
  - **HOW IS IT BEING DONE?**
    - » **I/O AND MEMORY REQUIREMENTS**
    - » **PARTICULAR ALGORITHMS (ACCESS METHODS)**
  - **TIME DEPENDENCIES AND FLUCTUATIONS**
  - **CONCURRENCY**
  - **IMPORTANCE**

Goal:

Characterize Load via a Uniform, Simple Procedure

LOAD PROFILES
WHY AND HOW

- **Necessary for Many Physical Design Decisions**
- **Load Profiles are Dynamic**
  - **AN ON-GOING PROCESS IN THE FACE OF LITTLE INFO**
  - **VARIANCE OR "RANDOM" FLUCTUATIONS**
  - **CYCLICAL (DAILY, WEEKLY, MONTHLY, YEARLY)**
  - **TREND (FOR EXAMPLE, GROWTH)**
- **Actual Load Profiles are often Complex**
  - **REDUCE COMPLEXITY BY REDUCING DETAIL**
- **Two Methods**
  - **MEASURE**
  - **MODEL**

LOAD PROFILES
MEASUREMENT

- **DBMS Instrumentation**
  - ALL ACCESS RESTRICTED TO STORED PROCEDURES
  - CREATE MULTIPLE LOGGING TABLES
  - INSERT TRACE VIA NESTED ASYNCH PROCEDURE
    - » ROLLUP VIA PERIODIC SWEEPING (CRON JOB, TRIGGER, etc.)
      - BEGIN: UPDATE SUM-STATS… WHERE MODULO(TIME) = CONSTANT: DELETE DETAIL-STATS: END
    - » ALTERNATIVELY ROLLUP VIA MERGE REPLICATION
  - WHAT ABOUT AD-HOC QUERIES?
    - » MOST FOLLOW A COMMON PATTERN
      - PARAMETERIZE AS STORED PROCEDURE AND SAVE
      - LET USERS SELECT BEFORE CREATING NEW QUERY
    - » AT LEAST JOINS AND CONDITIONS ARE COMMON
      - PRESENT AS “BUSINESS OBJECTS” AND LOG USE

LOAD PROFILES
MEASUREMENT

- **Monitoring or Trace Tools**
  - **SNIFFERS: *TELERAN* AND OTHERS**
- **Application Instrumentation**
  - **USER-WRITTEN**
    - » **LOG TO IN-MEMORY “FILE”**
    - » **COPY PERIODICALLY TO DATABASE**
  - **VENDOR-WRITTEN**
    - » **APPLICATION SERVER ENVIRONMENTS**
    - » **CHARGE-BACK ACCOUNTING**
    - » **TRACE / AUDIT FUNCTIONALITY**
- **Reduce Complexity via Sampling Where Possible**
  - **WHERE {conditions} AND RANDOM(min, max) /range) < sample-rate**

LOAD PROFILES
MODELING

Use Artificial Transactions That Represent the Typical Usage

❶ Identify Known Tasks

- DEFINE VIA TRANSACTIONS

❷ Prioritize (*P*) Transactions by Relative Business Importance

- USE A SIMPLE RANKING AND THEN NUMBER
- ALTERNATIVELY ASSIGN A SUBJECTIVE VALUE

❸ Quantify Transaction Volume (*V*)

- FREQUENCY, TIME DISTRIBUTION, I/O COST, ACCESS ORDER
  - » DATA ACCESS ORDER IS CRUCIAL FOR CONCURRENCY AND PARALLELISM
- ESTIMATE VARIANCE FOR EACH

LOAD PROFILES
MODELING

- **Modeling: Model Transactions**
  - CREATE A MODEL TRANSACTION FOR EACH REAL TRANSACTION
  - ELIMINATE REDUNDANT MODEL TRANSACTIONS
    - » MERGE TIME DISTRIBUTION AND VOLUME REQUIREMENTS OF ORIGINAL TRANSACTIONS
  - THESE CHARACTERIZE (OR *PROFILE*) THE LOAD
- **Model Databases: A Caution**
  - DON'T ASSUME RESULTS ON A "SCALED DOWN" DATABASE WILL SCALE UP!
    - » I/O AND BUFFER USE ARE NON-LINEAR
  - CALCULATED COSTS ARE USUALLY MORE ACCURATE

LOAD PROFILES
MODELING

- **Identify Access Patterns**
  - EACH TRANSACTION *X[k]* HAS QUERIES *Q[jk]* HAS PARTICIPATING TABLES *T[ijk]*
  - MEASURE OR ESTIMATE TRANSACTION VOLUMES *V[k]*
    - » AVERAGE, PEAK, DISTRIBUTION
    - » *NOTE:* CRUD AND VOLUMETRIC ANALYSIS ARE HELPFUL
  - WEIGHT TRANSACTIONS *X[k]* BY IMPORTANCE *P[k]*
- **Tune Individual Queries**
  - FOCUS ON MOST IMPORTANT QUERIES: TOP 20% *V[k] * P[k]*
  - USE THE "EXPLAIN" UTILITY ON EACH QUERY
    - » IDENTIFY BEST ACCESS METHODS *M[ijk]* AND "DRIVING" KEYS *K[ijk]* FOR EACH TABLE *T[ijk]* ACCESSED

    NOTE: THESE ARE CANDIDATE PARTITIONING KEYS FOR PARTICIPATING TABLES

WHERE WE ARE

- ✓ The Approach In Overview
- ✓ Scalability
- ✓ Data Independence
- ✓ Load Profiles
- Getting the Most Out of the Optimizer
- Denormalization
- Handling Missing Information
- Optimal Storage Management

PART IV

GETTING THE MOST OUT OF THE OPTIMIZER

APPLICATION PRINCIPLES

- **Set Processing**
  - AVOID ROW-AT-A-TIME LOOPS
    - » SELECT EMPNO FROM EMP
      - GET ARRAY OF EMPLOYEE NUMBERS
    - » BEGIN LOOP: CURNO = EMPNO[index]
    - » SELECT EMPNO, ESAL FROM EMP WHERE EMPNO = CURNO
    - » NEW-ESAL = ESAL * 1.1
    - » UPDATE EMP SET ESAL = NEW-ESAL WHERE EMNO = CURNO
    - » NEXT EMPNO
  - BETTER
    - » UPDATE EMP SET ESAL = ESAL * 1.1
- **Asynchronous Requests**

APPLICATION PRINCIPLES

- **Asynchronous Requests**
  - AVOID BLOCKING VIA BACKGROUND PROCESSING
    - » QUERY PROCESSING
    - » DATA RETRIEVAL
      - MOVE DATA IN BULK TO MEMORY OR A LOCAL FILE
    - » DATA MODIFICATIONS
      - MOVE DRIVING PARAMETERS IN BULK TO A WORK TABLE
      - PROCESS VIA A SET UPDATE, SET INSERT, OR SET DELETE
  - AVOID CONVERSATIONAL TRANSACTIONS
    - » RETRIEVE DATA ONLY IF NECESSARY
    - » AVOID “CONFIRMING” RETRIEVALS
      - YOU CAN STORE RESULTS IN AN AUDIT TABLE
      - YOU CAN ALWAYS RUN A REPORT LATER
    - » CONDITIONALIZE ALL TRANSACTIONS

UNDERSTANDING TRANSACTIONS

DESIGN ISSUES

- **Understand Transaction Structure**
  - AN INITIAL READ PHASE
  - AVOID RE-READING DATA
  - A WRITE PHASE BEGINS WITH THE FIRST MODIFICATION
    - » *INSERT, UPDATE, OR DELETE*
- **Minimize the Write Phase**
  - DATA TOUCHED
  - TIME TO COMMIT
  - CONSIDER PRE-READING DURING THE READ PHASE
    - » *IMPROVE CACHE HITS DURING WRITE PHASE*
- **Minimize Transaction Scope**
  - MINIMIZE NUMBER OF ACTIONS

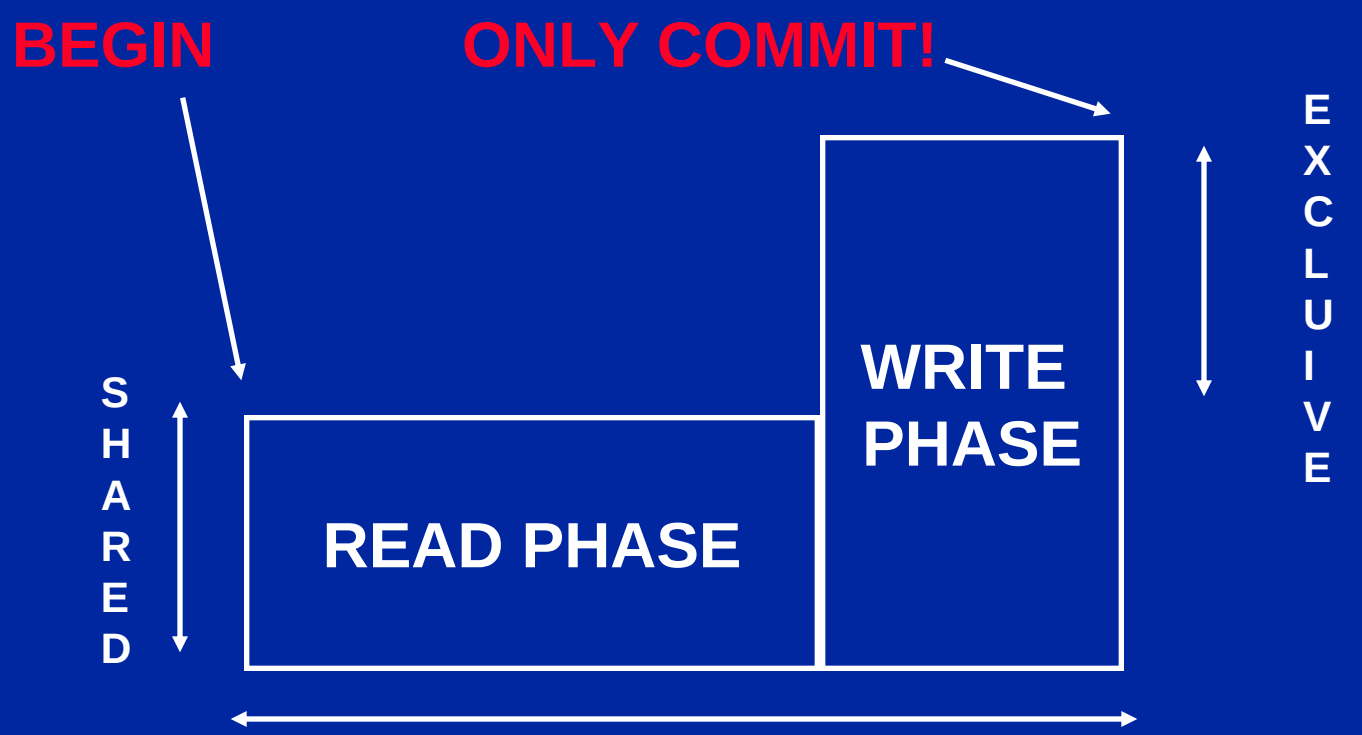
UNDERSTANDING TRANSACTIONS
DESIGN ISSUES

- **Commutative Property**
  - DEFINITION: ORDER INDEPENDENT
  - COMPLETE INDEPENDENCE OF STATE
- **Inverse Property**
  - PERMITS COMPENSATING TRANSACTIONS
    - » *a.k.a. "UNDO" TRANSACTIONS*
  - HELPS AVOID ROLLBACK
    - » SEE DO'S AND DON'TS
- **Ways to Avoid Long Running Transactions**
  - BOOKKEEPING
- **Execute Local to Any Necessary Shared Resources**
  - AVOID DISTRIBUTED TRANSACTIONS

UNDERSTANDING TRANSACTIONS
DO'S AND DON'TS

- **Follow Transaction Design Principles**
  - APPLIES TO ALL SHARED DATA RESOURCE ENVIRONMENTS
  - ***NOT*** JUST A DBMS ISSUE
  - JAVA OR ANY MULTI-THREADED APPLICATION
  - USE SERIALIZABLE TRANSACTIONS ONLY
- **Understand Your Codes Critical Sections!**
  - CRITICAL SECTION IF INTERRUPTION CAUSES CORRUPTION
- **Deadlock and Livelock**
  - CAN BE ACROSS SHARED RESOURCES OF ANY TYPE
    - » DATA, CPUs, DISKs, MEMORY, I/O, DISTRIBUTED COMPONENTS, . . .
- **Avoid Rollback**
  - VERY COSTLY AND CREATES RESOURCE CONTENTION
  - ROLLBACK ONLY ON UNAVOIDABLE PHYSICAL ERROR

UNDERSTANDING TRANSACTIONS
DESIGN ISSUES



TRANSACTION DESIGN
CONFLICT ANALYSIS

- **Identify Transactions That Can Interfere**

- **Why?**
  - **SCHEDULE TRANSACTIONS AND REDUCE CONTENTION**
    - » ***AVOID SUBMITTING TWO OR MORE TRANSACTIONS THAT REQUIRE LOCKING TO GUARANTEE ISOLATION***
    - » ***UNFORTUNATELY, YOU MUST DO THE SCHEDULING YOURSELF.***
  - **INCREASE RESPONSE TIME AND THROUGHPUT**

TRANSACTION DESIGN
CONFLICT ANALYSIS

Two Transactions Cannot Interfere If:

- **THEY DON'T TOUCH THE SAME DATA**
- **THEY ARE READ ONLY**
- **THEY COMMUTE**

OR

- **THEY DON'T RUN AT THE SAME TIME**

CONFLICT ANALYSIS
A DATABASE EXAMPLE

Which pairs of the following can interfere?

- ☆ **UPDATE SUPPLIERS SET SNAME = 'NEW_CO_NAME' WHERE SNAME = 'OLD_CO_NAME' AND CITY = 'NEW YORK'**
- **UPDATE SUPPLIERS SET SNAME = 'OLD_CO_NAME' WHERE SNAME = 'NEW_CO_NAME' AND CITY = 'NEW YORK'**
- **UPDATE SUPPLIERS SET SNAME = 'NEW_CO_NAME' WHERE SNAME = 'OLD_CO_NAME' AND CITY <> 'NEW YORK'**
- **UPDATE SUPPLIERS SET SNAME = 'NEW_CO_NAME' WHERE SNAME = 'OLD_CO_NAME' OR CITY <> 'NEW YORK'**

- **What level of transaction isolation enforcement is required?**
- **What is the effect of existence or non-existence of indexes?**

QUERY PRINCIPLES

- **Make Each Query Smart!**
- **Minimize Amount of Data Accessed**
- **Minimize Amount of Data Returned or Updated**
- **Divide and Conquer As Necessary**
  - **ASK FOR WHAT YOU NEED IN ONE QUERY**
    - » **PROVIDE ALL KNOWN COLUMN RELATIONSHIPS**
  - **FLATTEN SUBQUERIES**
  - **AVOID AGGREGATE FUNCTIONS**
  - **BREAK INTO ADDITIONAL QUERIES *ONLY AS NECESSARY***
  - **FINALLY, FORCE TEMPORARY DATA INTO WORK TABLES ONLY IF NECESSARY**

INDEXING PRINCIPLES

- **Avoid Table Scans**
  - **EVERY READ SUPPORTED BY AN INDEX**
  - ***EXCEPT FOR SMALL TABLES***
- ***Myth*: Indexes Slow Down Updates, Speed Up Reads**
  - ***REALITY*: SET UPDATES ALSO BENEFIT**
- **Concept of Simple Searchable Arguments**
  - **A SIMPLE BOOLEAN CONDITION WITH ONE OR MORE COLUMN REFERENCES**
  - **column <relationship> value**
  - **JOIN SSA: column <relationship> column**
  - **DISJUNCT SSA: SSA {OR SSA}... where all SSAs reference a common table.**

INDEXING PRINCIPLES

Problem:

"What is the minimum set of indexes that will cover the SSAs?"

- **FOR EVERY QUERY AND EVERY TABLE IN THAT QUERY, THERE SHOULD BE AT LEAST ONE SSA THAT REFERENCES A COLUMN OF THE TABLE AND IS INDEXED.**
- **IDEALLY, BOTH COLUMNS OF AT LEAST ONE JOIN SSA PER JOIN IN EACH QUERY SHOULD ALSO BE INDEXED**

INDEXING PRINCIPLES

Solution:

- IDENTIFY ALL TABLES AND FOR EACH TABLE A LIST OF QUERIES THAT REFERENCE THAT TABLE
- IDENTIFY ALL THE DISJUNCT SSAs PER QUERY
- FOR EACH TABLE
  - » FIND THE DISJUNCT SSA THAT APPEARS IN THE LARGEST NUMBER OF QUERIES AND INDEX IT
  - » REMOVE THE NOW INDEXED QUERIES FROM THE LIST
  - » IF NO QUERIES REMAIN, PROCEED TO THE NEXT TABLE
  - » OTHERWISE ITERATE FINDING NEXT MOST FREQUENT DISJUNCT SSA
- ITERATIVELY IMPROVE INDEXES
  - » ADD CO-APPEARING SSAs

INDEXING PRINCIPLES

- **Index Candidate Keys ... including Foreign Key Portions**
  - **DECLARING CONSTRAINTS MAY DO THIS FOR YOU**
- **B-Tree Index**
  - **RANGES, SPECIFIC VALUES, BETWEEN, LIKES WITH FIXED HEAD, ORDER MAY HELP ORDER BY, GROUP BY**
- **Bit Map Index**
  - **DISCRETE VALUES FROM DISCRETE DOMAINS**
  - **MAX. CARDINALITY DEPENDS ON PRODUCT**
- **Hybrid Bit Map**
  - **BIT MAP WITH RANGE CAPABILITY**
  - **LOWER STORAGE EFFICIENCY**

INDEXING PRINCIPLES

- **Hash Index**
  - DISCRETE VALUE (LISTS)
  - PARALLELISM
- **Function or Expression Index**
  - COMMON FUNCTIONS OR EXPRESSIONS IN SSAS
  - CAN SIMULATE VIA INDEX ON COMPUTED COLUMNS (TRIGGER)
- **Specialized Index**
  - SPECIFIC TO DATA TYPE
  - K-TREEE, R-TREE, T-TREE AND MANY MORE!
- **Join or (Multi-table) Index**
  - IDEAL FOR COMMON JOINS
  - “STAR INDEX” IS A SPECIAL CASE

INDEXING PRINCIPLES

- **Indexes and Parallelism**
  - PRACTICAL SIZE LIMITED BY BUILD TIME
  - PARTITIONING CAN HELP
    - » MAY HURT PARTIAL DATABASE RECOVERY
  - BUILD
    - » FASTER, BUT GENERALLY NO RESTART
  - SEARCH
    - » PARTITION AND PLACE ON A DIFFERENT DISK THAN DATA
  - UPDATE
    - » SAME AS SEARCH, BUT CONSIDER CONTENTION

WHERE WE ARE

- ✓ The Approach In Overview
- ✓ Scalability
- ✓ Data Independence
- ✓ Load Profiles
- ✓ Getting the Most Out of the Optimizer
- Denormalization
- Handling Missing Information
- Optimal Storage Management

PART V

DENORMALIZATION

PHYSICAL DATABASE DESIGN

- **The Design of Storage Structures**
  - **FOR PERFORMANCE**
  - **WITHOUT SUBVERTING RELATIONAL CORRECTNESS!**
  - **DON'T CONFUSE WITH DESIGN OF THE LOGICAL VIEW!**
- **Need Not Be Normalized If. . .**
  - **CAN HIDE PHYSICAL DEVIATIONS FROM FROM ALL USERS**
  - **ALL OPERATIONS MANIPULATE ONLY THAT LOGICAL VIEW**
  - **PHYSICAL SCHEMA UPDATES NEVER INDUCE LOGICAL ANOMALIES**

PHYSICAL DATABASE DESIGN

- **Method**
  - TREAT PHYSICAL SCHEMA AS A SET OF UPDATABLE VIEWS DEFINED FROM THE LOGICAL SCHEMA
    - » ***NOT*** THE REVERSE METHOD (AS IS MORE COMMON)!
  - ENFORCE PHYSICAL MULTI-TABLE CONSTRAINTS VIA TRIGGERS AND INTEGRITY CONSTRAINTS

Remember . . .

The Golden Guarantee of Data Independence

"ALL PHYSICAL COMPLEXITY CAN BE CONCEALED VIA ACCESS THROUGH THE LOGICAL SCHEMA"

PHYSICAL DATABASE DESIGN

- **With VLDB, Physical Design Rules Change**

  ***EXAMPLE*:**

  - » **COMPOUND KEYS IN VERY LARGE TABLES ARE OFTEN REDUNDANT, WASTING LOTS OF SPACE**

  ***SOLUTION*:**

  - » **REPLACE WITH SURROGATE KEYS AND A LOOKUP TABLE**

  ***EXAMPLE*:**

  - » **“FACT” TABLES OFTEN CONTAIN MULTIPLE ENTITIES WITH NULLABLE ATTRIBUTES**
  - » **CAUSES CONDITIONAL PROCESSING**

  ***SOLUTION*:**

  - » **NORMALIZE AND ELIMINATE NULLS**

"DENORMALIZATION"

- **Examples (Potentially Bad)**
  - JOINED TABLES
  - PARTITIONED AND REPLICATED TABLES
  - REDUNDANT COLUMNS
  - DERIVED COLUMNS
  - EMBEDDED FOREIGN KEYS
  - UNIONED ENTITIES (LEADS TO NULLS!)
  - various other reasons....
- **Why is this done?**
  - OPTIMIZING STORAGE ALLOCATION
  - MINIMIZING I/O COSTS, INCLUDING JOIN I/O
  - MAKING IT "EASIER" TO ACCESS RELATED INFORMATION

"DENORMALIZATION"*

- "Denormalization" (An Oxymoron!)
  - A PART OF THE PHYSICAL DATABASE DESIGN ONLY.
- What is Legitimate?
  - A SINGLE LOGICAL RELATION CAN BE REPRESENTED BY TWO OR MORE PHYSICAL TABLES
    - » JOIN, UNION, DIFFERENCE
  - MULTIPLE LOGICAL RELATIONS CAN BE REPRESENTED BY A SINGLE PHYSICAL TABLE
    - » PROJECTION, RESTRICTION

PHYSICAL DATABASE DESIGN
STRIPING AND RAID

STRIPING GOAL: ***Balance the I/O Load***

- RANDOM OR ROUND ROBIN SPREAD OF THE DATA
- ACROSS AVAILABLE CONTROLLERS AND DRIVES
- MAY BE DETRIMENTAL TO SEQUENTIAL AND RANGE SEARCHES

- **Multiple Tables Striped to One Drive or Controller**
  - CAN CAUSE CONTENTION
  - TREAT A DRIVE AS A SHARED "DATA" RESOURCE
    - » PERFORM A CONFLICT ANALYSIS
    - » BEST IF NO CONCURRENT USERS NEED THE SAME RESOURCE WITH RESOURCE > 40% CAPACITY
- ***RAID***
  - INTRODUCES LOSS OF PLACEMENT CONTROL

WHERE WE ARE

- ✓ The Approach In Overview
- ✓ Scalability
- ✓ Data Independence
- ✓ Load Profiles
- ✓ Getting the Most Out of the Optimizer
- ✓ Denormalization
- Handling Missing Information
- Optimal Storage Management

PART VI

HANDLING MISSING INFORMATION

WHEN ARE NULLS USED?

- **Conditional Data Entry**
  - USER (OR OTHER DATA SOURCE) HAS AN OPTIONAL DATA FIELD
- **Conditional Relationships**
  - SOME X'S ARE RELATED TO Y'S, BUT NOT ALL
- **Conditional Properties**
  - SOME X'S HAVE THE ATTRIBUTE BUT NOT ALL
- **Unidentified Entity Instances**
  - RELATIONSHIPS EXIST, BUT INSTANCE IS ABSTRACT
- **Conditional Operators**
  - THE OPERATOR IS NON-UNIFORM
  - PRODUCES NULLS TO FORCE UNIFORM OUTPUT

CONDITIONAL DATA ENTRY WITH DEFAULTS

(HANDLING MISSING INFORMATION)

Use for *Some* Conditional Data Entry, Namely:

- **When the Default Value Is**
  - **MEANINGFUL OR AN APPROPRIATE GUESS!**

  OR

  - **THE BEST ESTIMATE AND OTHERWISE HARMLESS**

  (*i.e.*., NOTHING DEPENDS ON THE PARTICULAR VALUE)

CRITICAL ASSUMPTION:

ALL SUCH DATA IS INTENDED TO BE IMPROVED UPON OVER TIME!

CONDITIONAL RELATIONSHIPS

(HANDLING MISSING INFORMATION)

EXAMPLE: Employee-managers

EMP-MGR (EMP#, ENAME, ESAL, MGR#)

- **AT LEAST ONE ROW CONTAINS A NULL FOR MGR#**
- **ALL EMPLOYEES ARE NOT OF THE SAME ENTITY TYPE!**

- **New approach: Create an associative relation**

  EMP (EMP#, ENAME, ESAL), MGR (MGR#, ...), M_E (EMP#, MGR#)

  - **ASYMMETRY PERMITS THE POSSIBILITY THAT SOME EMP# IS NOT MANAGED BY ANY MGR#**
  - **PHYSICALLY, I/O COST IS VERY LOW**
    - » **ESPECIALLY IF M_E IS COVERED BY AN INDEX AND GENERALLY CACHED**

CONDITIONAL RELATIONSHIPS

(HANDLING MISSING INFORMATION)

- **Recursive (Cyclic) Relations**
  - **IMPLY MULTIPLE ROLES ARE REPRESENTED IN A SINGLE ENTITY!**
  - **ASSOCIATION TABLES RESOLVE ANY N-CYCLE**
- **Associate Relations Can Model Any Relationship!**
- **Solves Referential Integrity Problems (“null” FKs)**
- **Conditional Relationships May Imply Subsetting**
  - **SEE CONDITIONAL PROPERTIES**

TYPES AND SUBTYPES

(HANDLING MISSING INFORMATION)

- **Logically, Each Subtype Is a Separate Relation**
  - REMOVING A COLUMN REPRESENTS GENERALIZATION OF THE TYPE
  - NOT THE SAME OPERATION AS PROJECTION
    - » MAKES NO STATEMENT ABOUT THE "MISSING" COLUMN!
  - CONVERSELY, A SUBTYPE IS A SPECIALIZATION
- **Eliminates Need for Conditional Operations**
  - OUTER JOIN, OUTER UNION, etc.
  - THESE CONFUSE GENERALIZATION AND PROJECTION
  - MULTIPLE SELECTS SCALE BETTER
    - » CONSIDER MULTIPLE STREAMS, INTERSPERSED FOR CERTAIN REPORT GENERATION TASKS
  - WILL NOT ADDRESS CONDITIONAL OPERATIONS FURTHER

CONDITIONAL PROPERTIES EXAMPLE

(HANDLING MISSING INFORMATION)

- **Suppose some employees are salaried and others are hourly**

  TRADITIONAL SCHEMA

  EMP(ENUM, ENAME, ESAL, ERATE)

  – **BOTH ESAL AND ERATE ARE NULLABLE, BUT BOTH MAY NOT BE NULL FOR ANY ROW**

  BETTER LOGICAL SCHEMA

  SALARIED_EMP(ENUM, ENAME, ESAL)

  HOURLY_EMP(ENUM, ENAME, ERATE)

- ***PROBLEM: MODELING TYPES AND SUBTYPES***

TYPES AND SUBTYPES

(PHYSICAL DESIGN)

Solution ☡: *One Physical Table With Two Views*

- DO THIS ONLY IF THE SUPERTYPE IS NEEDED
- DEFINE AN EXTENDED DOMAIN
  - » *SPECIAL VALUE TO INDICATE AN ILLEGAL SALARY OR ILLEGAL HOURLY RATE PHYSICALLY*
  - » *FOR THIS EXAMPLE, -1 IS NUMERIC AND EASILY EXCLUDED*
    - *NULLS DON'T WORK WELL!*
- ACCESS VIA PROJECTION / RESTRICTION VIEWS
  - » *NEVER LET THE APPLICATION SEE COLUMNS THAT DO NOT APPLY*
  - » *ELIMINATE NON-SALARIED ROWS FROM SALARIED_EMP*
  - » *ELIMINATE NON-HOURLY FROM HOURLY_EMP*

```
CREATE VIEW SALARIED_EMP AS SELECT ENUM, ENAME, ESAL FROM
  EMP WHERE  ERATE > -1
```

TYPES AND SUBTYPES

(PHYSICAL DESIGN)

Solution ⌘: *Two Physical Tables and One View*

- PRESUPPOSES ANY NEED TO MANIPULATE BOTH TABLES CAN BE DONE VIA A UNION VIEW
  - » *CREATE VIEW EMP AS SELECT ENUM, ENAME FROM SALARIED_EMP UNION SELECT ENUM, ENAME FROM HOURLY_EMP*
  - » *BE CAREFUL ABOUT UNION VIEW UPDATE SUPPORT!*
- NOTE THAT THE UNION VIEW REQUIRES CONVERTING EACH RELATION TO THE SUPERTYPE
- ELIMINATES THE NEED TO MANAGE ANY HIDDEN VALUES

- **Both Designs Improve User Understanding, Optimization, Storage Costs, I/O Costs**

UNIDENTIFIED ENTITY INSTANCES

(HANDLING MISSING INFORMATION)

Problem: **"The Unassigned Employee"**

CONSIDER CERTAIN NEW HIRES

- **ALWAYS REPORTS TO SOMEONE, PERHAPS FOR REASSIGNMENT**
- **ALWAYS RECEIVES PAYMENT AUTHORIZATION FROM SOMEONE**
- **CONCEPTUALLY BELONGS TO AN ABSTRACT OR VIRTUAL DEPARTMENT**
- **REPRESENTS FUNCTIONAL, ALBEIT ABSTRACT, BUSINESS ENTITY INSTANCE**

- **Often modeled with null for department "value"**

Solution:

Create a Value for the Abstract Department

WHERE WE ARE

- ✓ The Approach In Overview
- ✓ Scalability
- ✓ Data Independence
- ✓ Load Profiles
- ✓ Getting the Most Out of the Optimizer
- ✓ Denormalization
- ✓ Handling Missing Information
- • Optimal Storage Management

PART VII

OPTIMAL STORAGE MANAGEMENT

DIMENSIONAL SCHEMAS

FACTS, AND STARS AND (SNOW)FLAKES, OH MY!

DIMENSIONAL SCHEMAS
THE WRONG WAY

- **Star Join Depends on Cartesian Products**
  - FORMS CARTESIAN PRODUCT OF DIMENSION TABLES
  - CARTESIAN PRODUCTS DON'T SCALE
  - *STAR SCHEMAS DON'T SCALE*
    - » *TRUE EVEN WITH SPECIAL INDEXES AND JOIN ALGORITHMS*
  - FACT TABLES GROW FASTER THAN DIMENSION TABLES
    - » *CREATES AN I/O IMBALANCE OR SKEW*
    - » *REQUIRES A PHYSICAL RE-DESIGN*
- **Fact Tables Are Often Multi-entity (Multi-fact!)**
  - MANY COLUMNS ARE NULLABLE
  - OFTEN NO WELL-DEFINED PRIMARY KEY
  - FOREIGN KEYS OFTEN ADDED

DIMENSIONAL SCHEMAS
THE RIGHT WAY

- **Star Schema Designs Aren't Methodical**
  - NO CORRECTNESS TESTS NOR DESIGN PROCEDURES
  - EASY TO VIOLATE INTEGRITY
  - EASY TO OBTAIN NONSENSE "FACTS"
- **The Right Stuff**
  - LOGICAL DESIGN
    - » BUT IGNORE IRRELEVANT DEPENDENCIES
  - PHYSICAL DESIGN
    - » OPTIMIZE FOR MINIMUM I/O
    - » PHYSICAL TABLES MUST BE DERIVED FROM LOGICAL SCHEMA
      - NO LOSS OF INFORMATION OR DEPENDENCIES
    - » ***MOST IMPORTANT FOR A DATA WAREHOUSE!***

DIMENSIONAL SCHEMAS
THE RIGHT WAY

- ***Get the Benefits Without Abandoning Reason!***
  - **FULLY NORMALIZE THE LOGICAL DESIGN**
  - **USE ONLY THE DEPENDENCIES THAT MATTER TO THE APPLICATION - *RELATIVE NORMALIZATION***
    - » ***MANY DEPENDENCIES ARE NEVER SEEN BY THE APPLICATION***
    - » ***ATTRIBUTES MAY BE COMPLEX (A SET FOR A REPEATING GROUP) - BE CAREFUL!***
  - **OPTIMIZE THE PHYSICAL FOR MINIMUM STORAGE**
    - » ***HIGH SCAN COST OFTEN OUTWEIGHS JOIN COST***
  - **MAKE CERTAIN THE PHYSICAL IS COMPATIBLE WITH THE LOGICAL**

PRE-AGGREGATION AND SUMMARY TABLES

AGGREGATION

- Data Warehouses Often Require Aggregate Views
- Dynamic Aggregation is Too Expensive
- Precomputing Everything Is Too Expensive
  - MAY NEED MULTIPLE LEVELS OF AGGREGATION (BY DAY, WEEK, MONTH, YEAR)
  - MAY NEED MULTIPLE AGGREGATES (AVERAGE, SUM)
  - HORRIBLE FOR REFRESH OF A LARGE DATABASE
- Same Issues Apply To OLTP Aggregation

AGGREGATION

- **Hierarchies of Aggregation**
  - EACH LAYER IS A DERIVED, PARTIALLY STORED, DATABASE
  - DESIGN THE LAYER AS YOU WOULD ANY OTHER DATABASE
    - » EACH LAYER MUST FOLLOW DB DESIGN RULES
  - EACH LAYER SCHEMA DERIVED FROM THE PREVIOUS
  - REFRESH EACH LAYER FROM THE PREVIOUS
  - HIGHER AND HIGHER LEVELS OF ABSTRACTION
    - » BY PERIOD
    - » BY AGGREGATION GROUP
  - BOUNDARY RULE
    - » AN UPDATING TRANSACTION MUST NEVER UPDATE MORE THAN ONE LAYER!
      - CAN CREATE UNPREDICTABLE OUTCOMES
- **Multiple Hierarchies Are Possible**

AGGREGATION

- **Create a Derived, Partially Stored, Database**
  - **DESIGN THE LAYER AS YOU WOULD ANY OTHER DATABASE**
  - **STORE GREATEST COMMON DENOMINATOR OF THE AGGREGATES**
  - **CREATE VIEWS TO COMPUTE FINAL AGGREGATES DYNAMICALLY**
  - **EXAMPLE:**
    - » ***STORE SUM AND COUNT***
    - » ***CAN THEN DYNAMICALLY AND CHEAPLY COMPUTE SUM, AVERAGE, AND MEAN VIEWS, AMONG OTHERS***
  - **DON'T STORE ALL AGGREGATIONS**
  - **TECHNIQUE NOW USED BY IBM DB2 ("SUMMARY INDEXES")**

PARTITIONING

DATABASE PARTITIONING

- Partitioning A Database Into Multiple Groups of Tables
- Some Reason for Associating Tables
  - CONSISTENT DATA SETS
- Useful When Applications or Transactions Access Only One Database Partition at a Time
- Useful If the Use of a Table Is Associated With Some Physical Resource (e.g., Node or CPU)
  - MIGHT CREATE A HOT SPOT IF DONE INCORRECTLY
  - NOT THE SAME AS SCHEMA PARTITIONING WHICH IS GENERALLY USED TO CONTROL HOT SPOTS

USER/APPLICATION DATABASE PARTITIONING

- **Partitioning By User**
  - CLASSES OF USERS
  - COMMON AUTHORIZATIONS
  - COMMON TYPES OF TRANSACTIONS
- **Partitioning By Application**
  - COMMON COLLECTION OF TABLES OR TABLE SUBSETS
  - COMMON SET OF TRANSACTIONS
  - MAY REQUIRE RESYNCHRONIZATION OF TABLE COPIES
    - » Replication or batch copy management
    - » Batch integrity checks
    - » Work flow queue management

TABLE PARTITIONING

- Design Tools Don't Support
- Partitioning of Tables
- Based on Either Horizontal or Vertical Subsets
- Horizontal Subsets Can Be Specified in Many Ways
  - RANGE OR KEY: PARTITION# = f(KEY VALUE RANGE)
  - EXPRESSION: PARTITION# = f(EXPRESSION VALUE RANGE)
  - HASH AND RANDOM: PARTITION# = HASH(KEY VALUE)
  - SCHEMA: ASSIGN TABLE TO PARTITION
- Vertical Subsets Should Remove Contention and Reduce I/O
  - NON-LOSS ***PHYSICAL TABLE*** PROJECTIONS
  - LOCKING SHOULD BE ON THE ROW ***SUBSET*** (LOGICALLY)

TABLE PARTITIONING

- **Subset Partitioning and Subset Merging Should Be Online and Dynamic:**

- **No Vendor Supports Automatic *Most Frequently Used* Partitioning!**
  - **...YOU MAY BE ABLE TO SIMULATE**
- **User-Defined Partitioning Functions Should Be Permitted.**
- **Do Not Confuse Physical Partitioning Operations With Logical Database Operations.**

PARTITIONING SCHEMES
SELECTION

- **Compute Relative Value**
  - FOR A EACH TABLE *T[i]*
  - FOR EACH CANDIDATE PARTITIONING KEY *K[ijk]*, OVER ALL j AND k
  - WEIGHT (P) * TRANSACTION VOLUME (V) * NUMBER OF USES OF CANDIDATE (N)
  - *W[i] = V[i] * P[i] * N[i]*
  - IDENTIFY THE DISTRIBUTION OF CANDIDATE VALUES W
- **Apply the Significance Test**
  - A GOOD CANDIDATE FOR KEY PARTITIONING WILL BE VALUED AT LEAST TWO STANDARD DEVIATIONS ABOVE OTHERS
- **General Principle for Range Partitioning**
  - BEST FOR APPLICATIONS THAT USE TRANSACTIONS CONFINED TO A RANGE OF KEY VALUES

PARTITIONING SCHEMES
SELECTION

- **Identify Opportunities for Expression Parititioning**
  - INSTEAD OF SIMPLE KEYS
  - APPLY TO UNPARTITIONED TABLES
  - APPLY STEPS 1 AND 2 TO COMMON EXPRESSIONS
- **Identify Opportunities for *Hash and Random Partitioning***
  - APPLY TO UNPARTITIONED TABLES
  - HOT SPOTS TABLES
  - PARTITIONED HASH JOINS TABLES
  - PARTICULARLY USEFUL FOR OLTP UPDATE TRANSACTIONS
- **Use *Schema Partitioning* *Where Appropriate***
  - SMALL TO MEDIUM SIZE TABLES ACCESSED BY SCAN
  - TABLE ACCESSED WITH PARTITIONS ON A GIVEN NODE

PARTITIONS
SIZE DETERMINATION

Goal: ***Load Balance for I/O***

- ESSENTIALLY HORIZONTAL OR VERTICAL STRIPING
- NUMBER PARTITIONS = MIN(NUMBER CPUs, CONTROLLERS)

Solution:

- **Determine Relationship of Partition Key Value to I/O Distribution**
  - CRUD AND PREDICTED OR MEASURED VOLUMETRICS
  - USE CURVE FITTING TO OBTAIN A POLYNOMIAL FUNCTION f()
  - EST. I/O = f(KEY_VALUE)
  - KEY VALUE DISTRIBUTIONS DETERMINE I/O MULTIPLIER FUNCTION g(KEY_VALUE)
    - » DIVIDE BY ROWS PER BLOCK AND ROUND UP TO ESTIMATE I/O

PARTITIONS
SIZE DETERMINATION

- **Find I/O per partition**
  - SUMMATION OVER A RANGE OF VALUES IN PARTITION
  - FORMALLY ESTIMATE VIA INTEGRATION
    - » FIND TOTAL I/O OVER THE TIME PERIOD OF INTEREST (HOUR, DAY, WEEK, …)
      - THIS SHOULD INCLUDE ALL PEAKS
    - » DIVIDE BY MIN(CPUs, CONTROLLERS) TO DETERMINE I/O "SIZE" PER PARTITION
    - » I/O SIZE PER PARTITION = INTEGRAL FROM LOWER TO UPPER KEY VALUE BOUNDS
    - » FIRST PARTITION LOWER BOUND IS LOWEST KEY VALUE
    - » SOLVE FOR EACH CONSECUTIVE UPPER BOUND OF INTEGRATION IN TURN

PARTITIONS
SIZE DETERMINATION

Calculation Example

ASSUMPTIONS

f: EST. I/O = (KEY_VALUE)

g: MULTIPLIER = 3 * KEY_VALUE

KEY_VALUE RANGE = 0 - 1,000,000, 10 CONTROLLERS / CPUs

TOTAL I/O = 6,000,000

CALCULATION

$f * g = 3 * (KEY\_VALUE)^2$

INTEGRAL (f * g) = 6 * KEY_VALUE, INTEGRAL [a, b] = 6 * (b - a)

I/O PER PARTITION = 6,000,000 / 10 PARTITIONS = 600,000

1st PARTITION UPPER BOUND: 600,000 = 6 * b, SO b = 100,000

2nd PARTITION UPPER BOUND: 600,000 = 6 * (b - 100,000), SO b = 200,000

REPLICATION

REPLICATION

- Design Tools Don't Support
- A Combination of Controlled Redundancy and Partitioning
- Always Address Partitioning Design First
- Isolate and Treat "Local" Redundancy on a Per-partition Basis
- Treat "Distributed" Redundancy Last
- Insist on Parallel Replication

REPLICATION

- Generally Physical (Does Not Replicate SQL)
  - REQUIRES CARE IN ORDER OF APPLICATION
  - CAN CONSUME LARGE AMOUNT OF NETWORK I/O
  - CAN INCREASE LOGGING REQUIREMENTS
- Log Based Replication
  - MINIMIZES CONTENTION DURING UPDATES
  - INHERENTLY ASYNCHRONOUS
- Trigger Based Replication
  - TRIGGERS IN EXECUTION PATH LENGTH FOR ALL USERS
  - CAN BE SYNCHRONOUS OR ASYNCHRONOUS
- Copy Management
  - CAN OFFER BULK TRANSFERS FOR REFRESH

REPLICATION

- **Controlled Redundancy**
  - DEPENDS ON BEING ABLE TO SEPARATE READ FROM WRITE
- **Determine Number of "Read-only" Copies**
  - OPTIMIZE I/O LOAD ACROSS NODES

  GOALS:
  - MINIMIZE INTERNODE I/O AND HARDWARE CONTENTION
    - » *WITHOUT OVERRUNNING CACHE*

  METHOD:
  - *ASSIGN A COPY IF REFRESH I/O IS LOWER THAN REPLICA I/O*
- **Avoid Peer-peer Replication Except for Commutative Transactions**
  - THAT IS, ORDER SHOULD NOT MATTER (same answer, any order)

Questions?

BIOGRAPHY

David McGoveran is an industry analyst, and an international management and database consultant and president of Alternative Technologies (Boulder Creek, CA), specialists in solving difficult relational applications problems since 1981. He has authored numerous technical articles and co-authored several books (including those with Chris Date). His newest book is A Zero Management: Business Success in the New Millenium.

This seminar is based on his workshops: The Client/Server University: Designing Effective Databases, and Achieving Scalability

PLEASE FILL OUT YOUR EVALUATIONS...

Thank you!